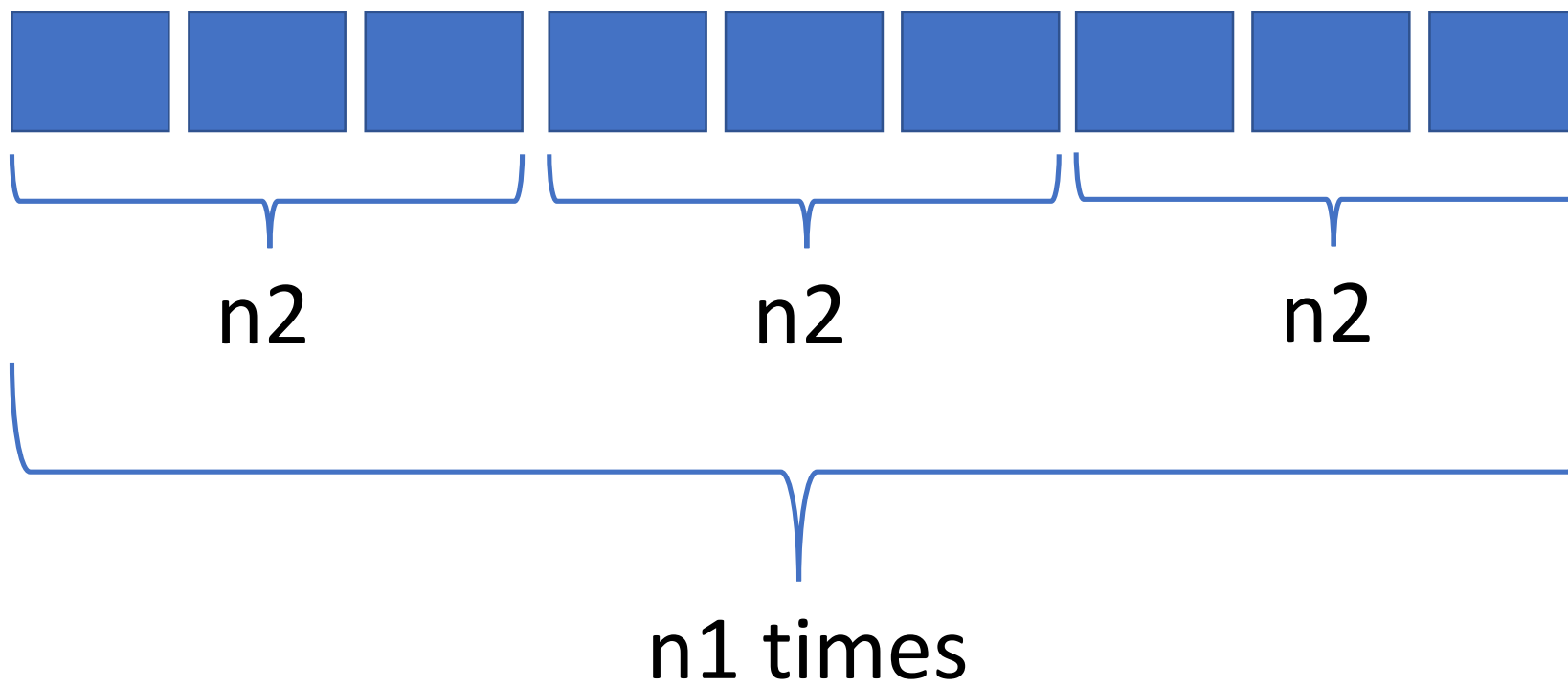


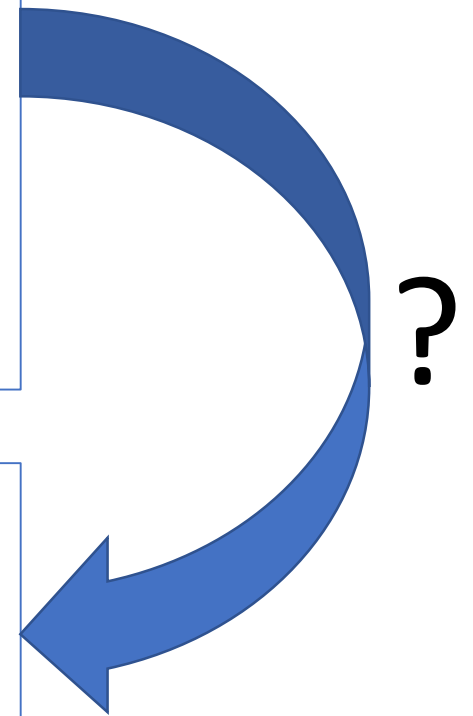
A story about vectorisation and compiler bug report

Arnaud Desitter
C++ On Sea – 16 July 2020



```
for (int i1 = 0; i1 < n1; ++i1)
{
    for (int i2 = 0; i2 < n2; ++i2)
    {
        res[i1*n2+i2] = a[i1*n2+i2] - b[i1*n2+i2];
    }
}
```

```
for (int index = 0; index < n1*n2; ++index)
{
    res[index] = a[index] - b[index];
}
```



With $n_1=100,000$ and $n_2=3$

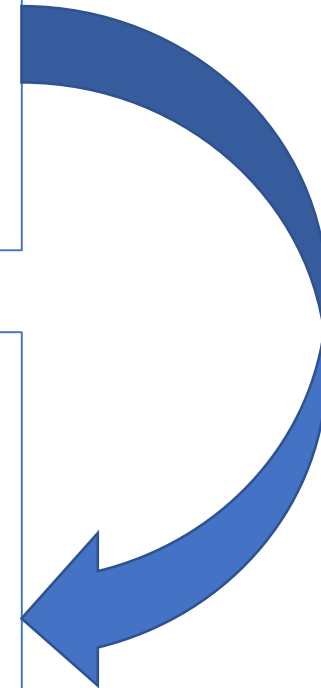
	penalty w.r.t. one flattened loop
gcc 9.1 -O3	+50%
gcc 9.1 -O2	+24%
clang 8.0 -O3	+34%
clang 8.0 -O2	+38%

Compilers only vectorise the inner loop.

Let's try OpenMP

```
for (int i1 = 0; i1 < n1; ++i1)
{
    for (int i2 = 0; i2 < n2; ++i2)
    {
        res[i1*n2+i2] = a[i1*n2+i2] - b[i1*n2+i2];
    }
}
```

```
#pragma omp simd collapse(2)
for (int i1 = 0; i1 < n1; ++i1)
{
    for (int i2 = 0; i2 < n2; ++i2)
    {
        res[i1*n1+i2] = a[i1*n1+i2] - b[i1*n1+i2];
    }
}
```



With $n_1=100,000$ and $n_2=3$

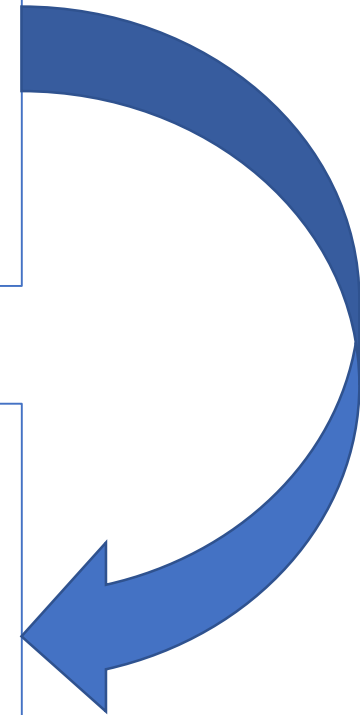
	penalty w.r.t. one flattened loop
gcc 9.1 -fopenmp-simd -O3	+100% (1)
clang 7.0 -fopenmp-simd -O3	+1230%
clang 8.0 -fopenmp-simd -O3	+0%

(1) https://gcc.gnu.org/bugzilla/show_bug.cgi?id=89371

Let's try to collapse the loops manually

```
for (int i1 = 0; i1 < n1; ++i1)
{ for (int i2 = 0; i2 < n2; ++i2)
  {
    res[i1*n2+i2] = a[i1*n2+i2] - b[i1*n2+i2];
  }
}
```

```
for (int index = 0; index < n1*n2; ++index)
{
  int i1 = index / n2;
  int i2 = index % n2;
  res[i1*n2+i2] = a[i1*n2+i2] - b[i1*n2+i2];
}
```



```
(index / n2) * n2 + index % n2 == index
```


With $n_1=100,000$ and $n_2=3$

	penalty w.r.t. one flattened loop
gcc 9.1 -O3	+383%
clang 8.0 -O3	+0%



Add... ▾

More ▾

Share ▾

Other ▾

Policies ▾

C++ source #1 ×



A ▾

Save/Load

+ Add new... ▾

CppInsights

C++ ▾

```
1 int f(int index, int stride) {  
2     const int i1 = index / stride;  
3     const int i2 = index % stride;  
4     const int index2 = i1*stride+i2;  
5     return index2; // expect: index2 = index  
6 }
```

x86-64 clang 8.0.0 (Editor #1, Compiler #1) C++ ×



x86-64 clang 8.0.0 ▾



-O3 ▾

A ▾

☐ 11010

☒ .LX0:

☐ lib.f:

☒ .text

☒ //

☐ \s+

☒ Intel

☒ Dema

Libraries ▾

+ Add new... ▾

⚙ Add tool... ▾

```
1 f(int, int):
```

```
2     mov     eax, edi
```

```
3     ret
```



Output (0/0)

x86-64 clang 8.0.0



- cached (11449B)



Add... ▾

More ▾

Share ▾

Other ▾

Policies ▾

C++ source #1 ×

A ▾

Save/Load

+ Add new... ▾

CppInsights

C++ ▾

```
1 int f(int index, int stride) {  
2     const int i1 = index / stride;  
3     const int i2 = index % stride;  
4     const int index2 = i1*stride+i2;  
5     return index2; // expect: index2 = index  
6 }
```

x64 msvc v19.14 (WINE) (Editor #1, Compiler #1) C++ ×

x64 msvc v19.14 (WINE) ▾



/O2 ▾

A ▾

☐ 11010

☒ .LX0:

☐ lib.f:

☒ .text

☒ //

☐ \s+

☒ Intel

☒ Dem

Libraries ▾

+ Add new... ▾

⚙ Add tool... ▾

```
1 index$ = 8  
2 stride$ = 16  
3 int f(int,int) PROC  
4     mov     eax, ecx  
5     ret     0  
6 int f(int,int) ENDP
```



Output (1/0)

x64 msvc v19.14 (WINE)



- cached (470B)

Add... ▾

More ▾

Share ▾

Other ▾

Policies ▾

C++ source #1 ×



A ▾

Save/Load

+ Add new... ▾

CppInsights

C++ ▾

```
1 int f(int index, int stride) {  
2     const int i1 = index / stride;  
3     const int i2 = index % stride;  
4     const int index2 = i1*stride+i2;  
5     return index2; // expect: index2 = index  
6 }
```

x86-64 gcc 8.3 (Editor #1, Compiler #1) C++ ×



x86-64 gcc 8.3 ▾



-O3 ▾

A ▾

☐ 11010☒ .LX0:☐ lib.f:☒ .text☒ //☐ \s+☒ Intel☒ Dema

Libraries ▾

+ Add new... ▾

Add tool... ▾

```
1 f(int, int):  
2     mov     eax, edi  
3     cdq  
4     idiv    esi  
5     imul    eax, esi  
6     add     eax, edx  
7     ret
```



Output (0/0) x86-64 gcc 8.3 ⓘ - cached (4768B)

Bug 89518 - missed optimisation for array address calculations

Arnaud Desitter

2019-02-27 11:41:08 UTC

[Description](#)

Considering:

```
int f(int index, int stride) {  
    const int i1 = index / stride;  
    const int i2 = index % stride;  
    const int index2 = i1*stride+i2;  
    return index2; // expect: index2 = index  
}
```

gcc 8.3 with "-O3" on x84_64 emits:

```
f(int, int):  
    mov     eax, edi  
    cdq  
    idiv    esi  
    imul    eax, esi  
    add     eax, edx  
    ret
```

By contrast, clang 7 with "-O3" emits

```
f(int, int):  
    mov     eax, edi  
    ret
```

MSVC 2017 with "/O2" emits:

```
int f(int,int)  
    mov     eax, ecx  
    ret     0
```

Is there a way to persuade gcc to simplify this expression at compile time?

Richard Biener

2019-02-27 11:54:44 UTC

[Comment 1](#)

We do not have a $(a / b) * b + (a \% b)$ simplification rule. The following adds one:

Index: gcc/match.pd

```
=====
--- gcc/match.pd          (revision 269242)
+++ gcc/match.pd          (working copy)
@@ -2729,6 +2729,13 @@ (define_operator_list COND_TERNARY
    (mult (convert1? (exact_div @0 @1)) (convert2? @1))
    (convert @0))

+/* Simplify (A / B) * B + (A % B) -> A. */
+(for div (trunc_div ceil_div floor_div round_div)
+  mod (trunc_mod ceil_mod floor_mod round_mod)
+  (simplify
+   (plus:c (mult:c (div @0 @1) @1) (mod @0 @1))
+   @0))
+
+/* ((X /[ex] A) +- B) * A --> X +- A * B. */
+(for op (plus minus)
+  (simplify
```

Richard Biener 2019-02-27 11:54:44 UTC

[Comment 1](#)

We do not have a $(a / b) * b + (a \% b)$ simplification rule. The following adds one:

Index: gcc/match.pd

```
=====
--- gcc/match.pd          (revision 269242)
+++ gcc/match.pd          (working copy)
@@ -2729,6 +2729,13 @@ (define_operator_list COND_TERNARY
    (mult (convert1? (exact_div @0 @1)) (convert2? @1))
    (convert @0))

+/* Simplify  $(A / B) * B + (A \% B) \rightarrow A$ 
+(for div (trunc_div ceil_div floor_div round_div)
+  mod (trunc_mod ceil_mod floor_mod round_mod)
+  (simplify
+    (plus:c (mult:c (div @0 @1) @1) (mod @0 @1))
+    @0))
+
+/*  $((X / [ex] A) +- B) * A \rightarrow X +- A * B$  */
(for op (plus minus)
  (simplify
```

13 minutes.
Thank you !



Add...

More

Do you have any suggestions,
requests or bug reports? Feel free to
[contact us](#) at anytime

Sponsors

PC-lint

PVS-Studio

Solid Sands

Share

Other

Policies

C++ source #1

A B + v 🔍

C++

```
1 int f(int index, int stride) {  
2     const int i1 = index / stride;  
3     const int i2 = index % stride;  
4     const int index2 = i1* stride+i2;  
5     return index2; // expect index2 = index  
6 }
```

x86-64 gcc 10.1 (Editor #1, Compiler #1) C++

x86-64 gcc 10.1



-O3

A Output... Filter... Libraries + Add new... Add tool...

```
1 f(int, int):  
2     mov     eax, edi  
3     ret
```

Output (0/0) x86-64 gcc 10.1 - 379ms (4353B)

With $n_1=100,000$ and $n_2=3$

	penalty w.r.t. one flattened loop
gcc 9.1 -O3	+383%
gcc 10.1 -O3	+0%
clang 8.0 -O3	+0%

Reporting bugs pays off